MSA220 - Statistical Learning for Big Data

Review Lecture

Rebecka Jörnsten

Mathematical Sciences
University of Gothenburg and Chalmers University of Technology

Topics

- Supervised Learning/Classification
- Unsupervised Learning/Clustering
- Dimension Reduction/Feature selection

Classification:

Goals and setup:

- Build a predictive model/algorithm to discriminate between known groups
- Need a data set with know group labels on which we can construct/learn our model.

Combine with dimension reduction/feature selection:

- increased interpretability - which features discriminate between the groups?
- improved performance - don't train your model on features that are unrelated to the groups.
- some methods can't be applied in a high-dimensional setting so you need to reduce the number of features first.

Clustering:
Goals and setup:

- Finding groups in data
- Summarize data this is otherwise difficult to "get a feeling for".
- Data exploration

This is a more difficult task than classification in the sense that the goal is *subjective*. What is a group? A set of observations separated from the rest? A set of observations close together? What is meant by "close"?

Clustering:
Combine with dimension reduction/feature selection:

- Which features are "responsible" for group separation?
- Some methods can't be applied in a high-dimensional setting so you need to reduce the number of features first. In a very high-dimensional setting "closeness" looses meaning (curse of dimensionality).

Classification - selection of tuning parameters

Classification methods can be tuned via method-specific parameters.

- kNN - choose the size neighborhood, k
- CART - choose number of rectangular regions or, equivalently, the number of data splits in the classification tree.
- Number of features
- LDA vs QDA vs FDA - complexity

How do we choose the best value for the tuning parameter? Cross-validation!!!

Prediction models

- y is the output, an $n \times 1$ vector
- X is the input (explanatories, independent variables....), an $n \times p$ matrix
- We want to build a predictive model or rule for y using X

What is a good model?

- Need to define a loss function to define "good"
- Usually squared error loss if y is continuous
- Usually 0-1 error loss if y is categorical

Predictive models

- $E(y - f(X))^2 = E_X[E_{y|X}[(y - f(X))^2 \mid X]]$
- We see that it is enough to minimize the above *pointwise* (for each $X = x$) so we focus on the inner expression
- $f(x) = \arg\min E_{y|X}[(y - f(x))^2 \mid X = x]$
- The minimizer is the *conditional mean* $\widehat{f(x)} = E(y \mid X = x)$
- For 0-1 loss the minimizer is the maximum probability class $\arg\max_c P(y = c \mid X = x)$.

The above conditional mean is called the *regression function*. It is the model that minimizes squared error loss.

How do we estimate $E(y \mid X = x)$?

- An intuitive approach is the approximate the conditional mean by the *local mean* defined by a neighborhood of observations $N(x)$ (e.g. k-nearest neighbors in X to observation x).
- Alternatively, we can parameterize the conditional mean via e.g. a linear model $E(y \mid X = x) = x'\beta$ or some other parametric form.

Bias-Variance Trade-off

How we choose to estimate the conditional mean determines how flexible/local or rigid/global we are in our modeling. Most methods allow us to choose from a spectrum of more or less local rules. We have already discussed that one needs to select tuning parameters for classification rules, e.g. the number of neighbors for kNN.

Depending on the tuning parameter value, a classification rule can be thought of as *local* or *global*.

local	global
use subset of data	use all data
flexible	more rigid
allow for complex boundaries or models	assume an underlying model for the data distribution
example: kNN with small k	example: discriminant analysis (multivariate normal data distribution)
example: Local average regression	example: linear regression model
need a lot of data to train on	requires less data in general

Nearest centroid classifier

A simple rule that is related to discriminant analysis is the nearest centroid classifier.
We compute

$$\hat{\mu}_c = \frac{1}{N_c} \sum_{y_i=c} x_i$$

where N_c is the number of observations in class c. That is, we compute the mean, or centroid, of each class.
The rule is

$$\hat{c}(x) = \arg\min_c d(x, \hat{\mu}_c)$$

where $d(.,.)$ is the distance between observation location x and the centroid μ. This is usually the euclidean distance

$$d(x, \hat{\mu}_c) = || \, x - \hat{\mu}_c \, ||^2 = (x - \hat{\mu}_c)'(x - \hat{\mu}_c).$$

The rule is thus to allocate each observation to the class with the closest centroid.

Nearest centroid classifier

Problems?
Nearest centroids ignores the variability of a class around its center and that this variability may be different for different classes and for different features.
One needs to consider both *spread/scale* of a distribution (the amount of spread around a centroid) and the *shape* of the distribution (the correlation structure between the features) to form a good classification rule.
This is what discriminant analysis does.

General setup is the following;

- prior $\pi_c = p(y = c)$
- data distribution $p(x \mid y = c) \sim N(\mu_c, \Sigma_c)$ where μ_c is a p-dimensional vector and Σ_c is a p-by-p dimensional covariance matrix.

The multivariate normal assumption leads to the following simple, intuitive parameter estimates:

- $\hat{\pi}_c = N_c/N$, where $N_c = \sum_i 1\{y_i = c\}$ is the number of observations in class c.
- $\hat{\mu}_c = \frac{1}{N_c} \sum_{y_i=c} x_i$
- $\hat{\Sigma}_c = \sum_{y_i=c} (x_i - \hat{\mu}_c)(x_i - \hat{\mu}_c)'/(N_c - 1)$

This is quite a large number of parameters...: $(C-1)$ for $\hat{\pi}$ (not C since the πs add to 1), $p \times C$ mean parameters, and $p(p+1) \times C/2$ covariance parameters (since they're symmetric). As the dimensionality of the problem grows (p) the number of parameters grows quickly, especially due to the covariance matrices.

Cautionary remarks

One take-home message is that a key component in LDA is the inverse of Σ, the within-class covariance matrix. This is also a main ingredient in regression type methods!
Problem?

- Σ may be very difficult to invert, numerically unstable, if the sample size n is not much bigger than the data dimension p.
- If some of the x-features are highly correlated, then the matrix Σ is also difficult to invert since it is near singular.
- Special case when $n < p$ or some x-s are perfectly correlated, the inverse of Σ does not exist. This is less of a worry since most programs will warn you about this. When we are *near* singular, that's when you need to pay attention. So correlated xs, high-dimensional data and small sample sizes are all situations when LDA and regression can fail due to the poor performance of the inverse of Σ.

Recap on regression

We want to fit a regression model to our data using least squares.

$$y = X\beta + \epsilon$$

- y is our $n \times 1$ vector of outcome data
- X is our $n \times p$ design matrix
- ϵ is our $n \times 1$ vector with additive errors.
- For convenience, assume centered and standardized data.

Is this OK?
Reality check: 5 basic assumptions, scatter plots,....
TRANSFORMATIONS! ID EXTREME OUTLIERS!

When p is large or covariates in X are correlated, it is a tricky business to fit regression via OLS.
Why?

- $\min_\beta ||y - X\beta||^2$ has closed form solution
- $(X'X)^{-1}X'y$
- IF the inverse of $X'X$ exists.
- Not true if $p > n$. Inverse unstable if some covariates extremely correlated.

Regularized regression

What do we do?

- Reduce the number of covariates - prefiltering
- PCA of X and use only leading components.
- Regularized regression

Ridge regression

We want to minimize

$$||y - X\beta||^2$$

$$\text{subject to } ||\beta||_2^2 \leq \tau$$

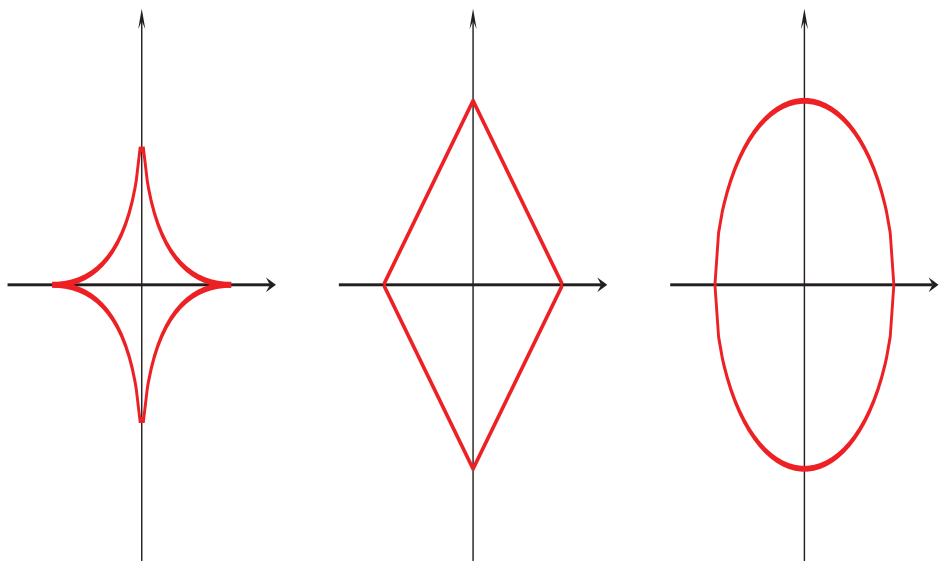
I.e., try to minimize least squares but don't let the average β get too big...

- Lagrangian formulation: $\min_\beta \frac{1}{2}||y - X\beta||^2 + \lambda||\beta||_2^2$
- Take derivatives with respect to β
- $-X'(y - X\beta) + \lambda\beta = 0$
- Solution $\beta_R = (X'X + \lambda I)^{-1}X'y$
- Choose λ to make sure condition τ holds or more commonly, choose λ via Cross-validation

Lq-penalized regression

We can adress the instability of high-dimensional modeling using different kinds of penalties.

- L_0: $\min_\beta ||y - X\beta||^2 + \lambda \sum_{j=1}^p 1\{\beta_j \neq 0\}$
- L_q: $\min_\beta ||y - X\beta||^2 + \lambda \sum_{j=1}^p |\beta_j|^q$
- L_1: $\min_\beta ||y - X\beta||^2 + \lambda \sum_{j=1}^p |\beta_j|$
- L_1: $\min_\beta ||y - X\beta||^2 + \lambda \sum_{j=1}^p \beta_j^2$



Fraction $q < 1$, $q = 1$ and $q = 2$

Because the $L1$ penalty has "singularities" (points) this makes the selection of solutions at those points more likely.
We will see this by solving the problem mathematically too, but think of this as the penalty region extremes being the most likely to lead to a solution that is optimal for the loss (model fit).

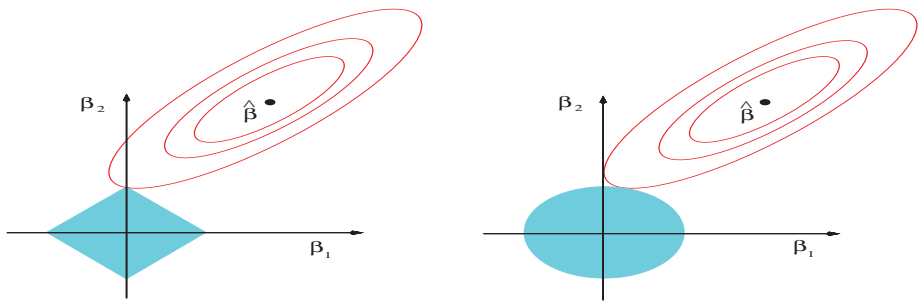
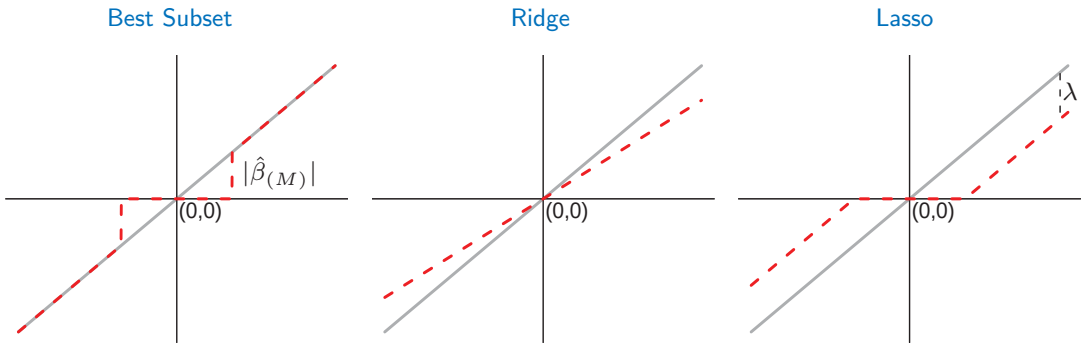


Figure 3.12: *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$, *respectively, while the red ellipses are the contours of the least squares error function.*

L1-penalized regression

Estimator	Formula
Best subset (size M)	$\hat{\beta}_j \cdot I(\lvert\hat{\beta}_j\rvert \geq \lvert\hat{\beta}_{(M)}\rvert)$
Ridge	$\hat{\beta}_j/(1+\lambda)$
Lasso	$\text{sign}(\hat{\beta}_j)(\lvert\hat{\beta}_j\rvert - \lambda)_+$

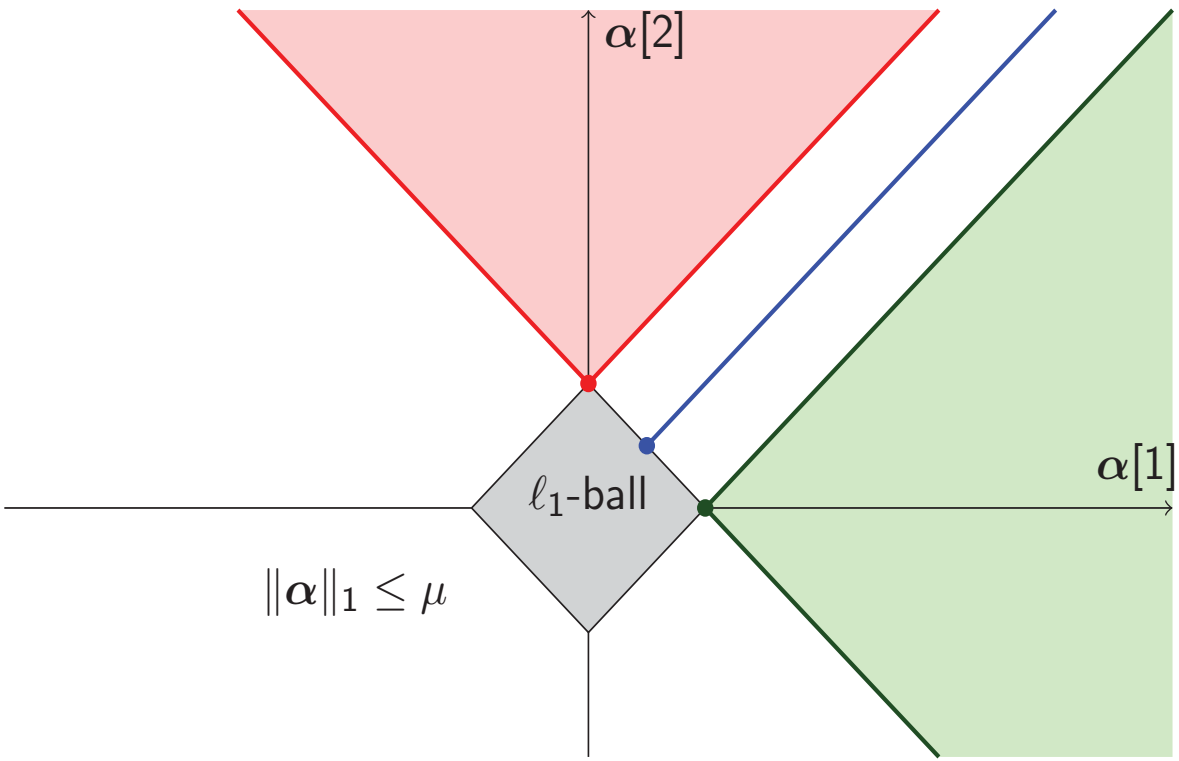


This figure from the book illustrates the estimation of $L0, L1$ and $L2$ in relation to the LS estimate.
Notice the constant bias for $L1$ and growing bias for $L2$.

So now you've seen that the L1-penalty induces model selection or *sparsity*. The reason is the non-smooth shape of the penalty region. Here follows some illustration from Julien Mairal's excellent slides.

Why does the ℓ_1-norm induce sparsity?

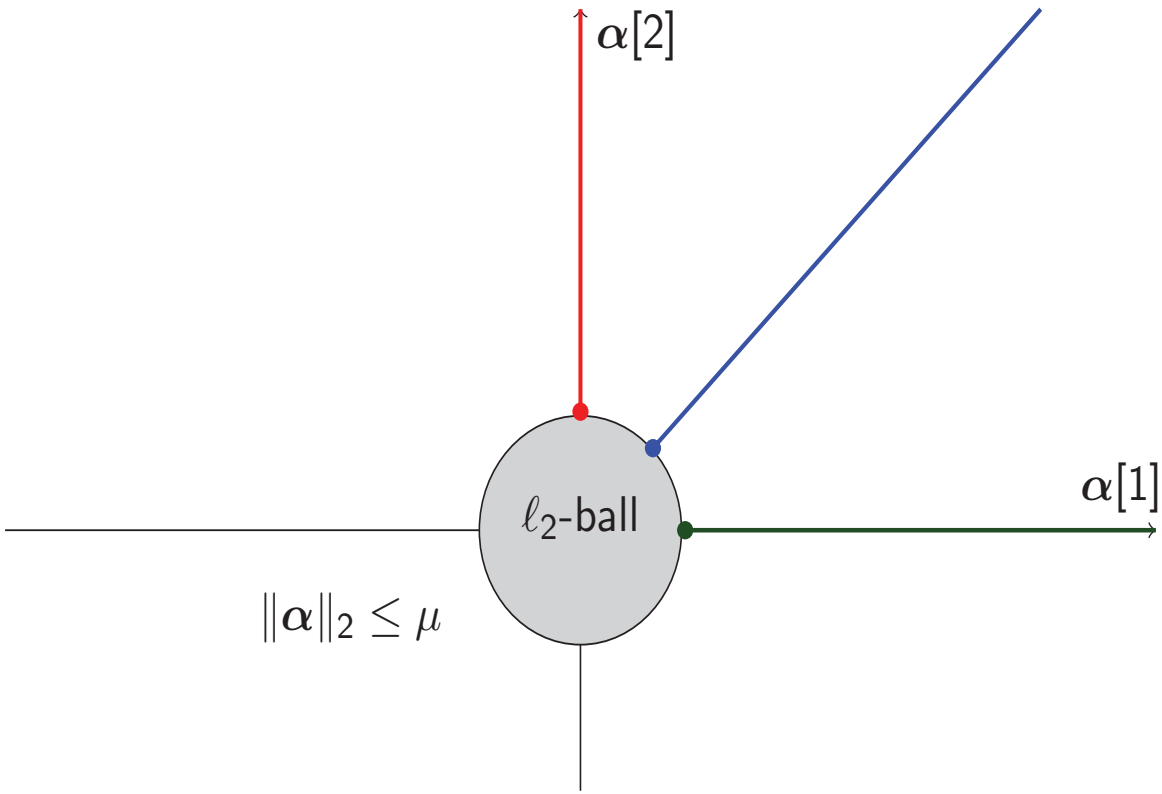
Regularizing with the ℓ_1-norm



The projection onto a convex set is "biased" towards singularities.

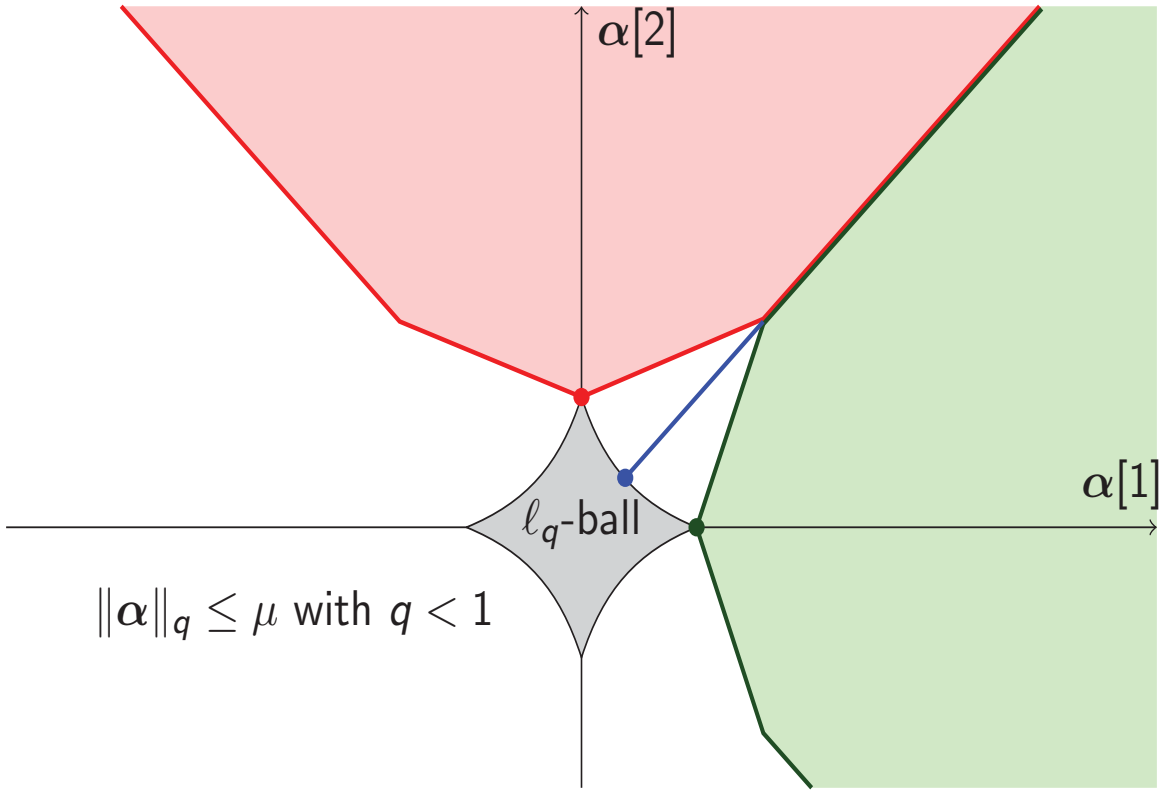
Why does the ℓ_1-norm induce sparsity?

Regularizing with the ℓ_2-norm



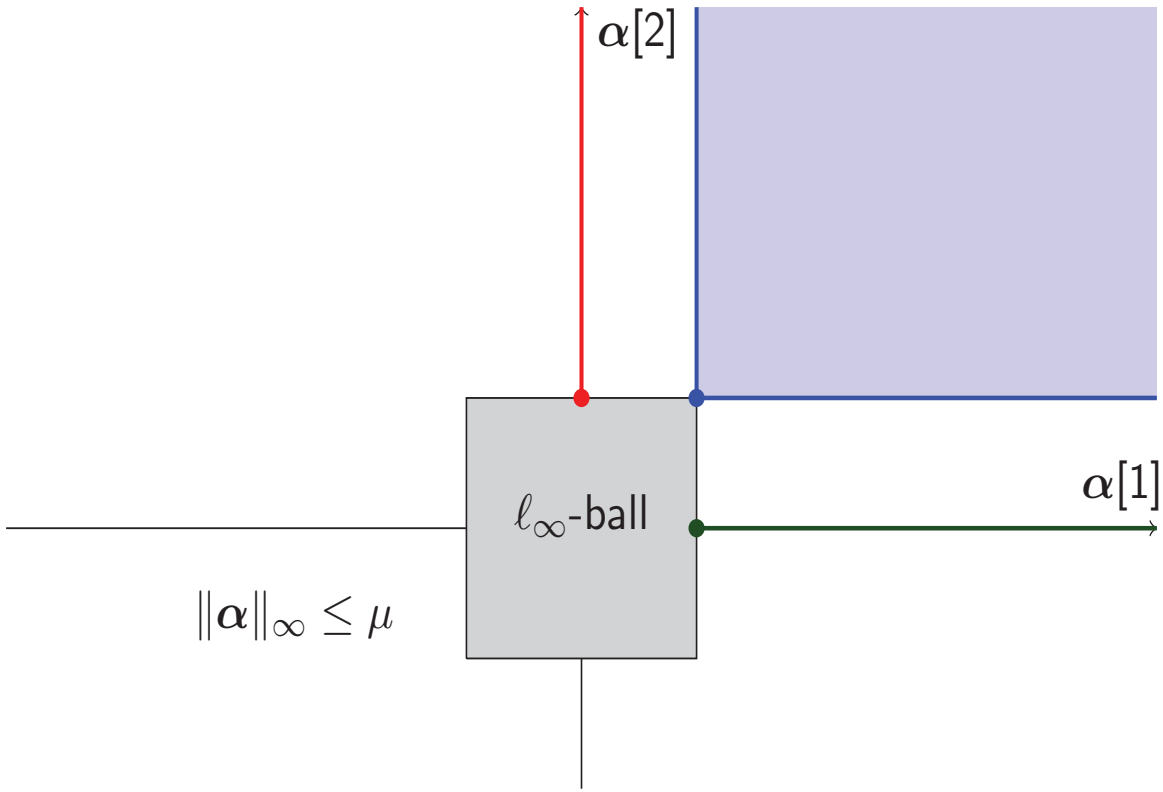
The ℓ_2-norm is isotropic.

Non-convex sparsity-inducing penalties



Why does the ℓ_1-norm induce sparsity?

Regularizing with the ℓ_∞-norm



The ℓ_∞-norm encourages $|\boldsymbol{\alpha}[1]| = |\boldsymbol{\alpha}[2]|$.

The point is that we can be a bit more adventurous about constructing penalty regions in order to generate a desired type of sparsity.

Group-penalized regression

Let's say there's a natural grouping of the variables: e.g. factor levels of a categorical level, source, etc. We want to include a group of variables and not select them separately.
We can achieve this by using a *group penalty* instead. Consider the case of K groups of variables:

$$\min_{\beta} \frac{1}{2}||y - X\beta||^2 + \lambda \sum_{k=1}^{K} ||\beta_k||_2$$

where $||\beta_k||_2 = \sqrt{\sum_{j \in k} |\beta_j|^2}$, i.e. we penalize the average β value within each group.

Group LASSO Ball

- If $D_k \in \mathbb{R}^{N \times 1}$, then $||X||_{2,1} = ||X||_1$
- That is, if all the groups are singletons, the optimization problem reduces to LASSO.
- Group LASSO ball shares attributes of both ℓ_2 and ℓ_1 balls.

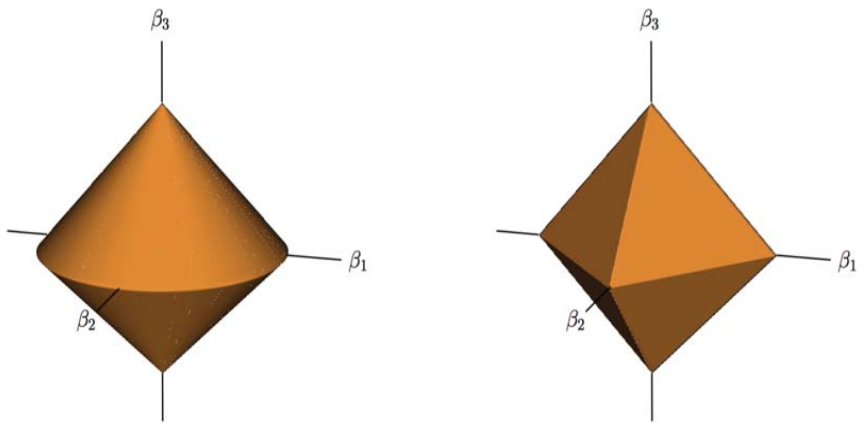


Figure 2: Group LASSO LASSO

From a talk by Hojjat Akhondi-Asl, UCL.

Elastic net

Lasso struggles when covariates are correlated and tends to pick only one of them even if both are related to the outcome. We can form groups of correlated variables and run group-lasso (Howard Bondell and others) or we can let the data decide for us and "helping" a bit by altering the penalty as follows:

$$\min_{\beta} \frac{1}{2}||y - X\beta||^2 + (1-\alpha)\lambda\frac{1}{2}||\beta||_2^2 + \alpha\lambda||\beta||_1$$

As you can see, this uses both an $L1$ and an $L2$ penalty on β. This penalty strategy is called the *elastic net*.

Summary

- Structured penalty to induce desired sparsity
- Simple coordinate descent works for big problems but perhaps not the most efficient
- Cross-validation usually used to select the penalty parameters, but this is a difficult problem.

Summary

How well do the L1-penalized methods perform?

- Biased estimates $\rightarrow$ *adaptive lasso, SCAD*
- If $\lambda = o(n)$, then $\beta_{l1-pen} \rightarrow \beta_{true}$ as $n \rightarrow \infty$
- If $\lambda \propto n^{1/2}$ L1-pen has a non-zero probability of identifying the true model (model selection consistency) (Knight and Fu, 2000)
- BUT if many of the non-relevant variables are correlated with the relevant variables, L1-pen regression may fail to select the true model even if n is large.
- We need the *Irrepresentable condition* to hold

$$|(X_1'X_1)^{-1}(X_2'X_2)| < 1 - \eta$$

where X_1 are the irrelevant and X_2 the relevant variables. (Zhao and Yu, 2006)

Generalized linear models

When we have categorical outcome data we can still use penalized fitting strategies.
The objective function will now be the negative log-likelihood of the data plus the penalty.
For binomial data, our model is

$$P(y = 1|X = x) = \frac{e^{x\beta}}{1 + e^{x\beta}}$$

and so we minimize

$$\sum_i (y_i(x\beta) - log(1 + e^{x\beta})) + J(\beta)$$

- This is nonlinear in β
- Use a quadratic approximation of the log-likelihood and coordinate descent
- Then the problem looks like a penalized least squares problem
- Solve this and iterate
- package `glmnet()`

What if you have more than two classes? Multinomial model.

$$P(y = k|X = x) = \frac{e^{x\beta_k}}{\sum_{j=1}^{K} e^{x\beta_j}}$$

- Notice we actually have set of coefficients $\beta_k = \{\beta_{0k}, \beta_{1k}, \cdots, \beta_{jk}\}$, one p-vector for each class.
- This $p \times K$ matrix of coefficients can be treated as separate problems
- OR if you want to have a model that is easier to interpret you let each $\beta_{jk}, k = 1, \cdots, K$ constitute a *group* so that variable j is used to predict all classes or not used at all.
- package `glmnet()`

High-dimensional inference

- L1-penalized modeling has become enormously popular for high-dimensional problems
- We get model selection, fairly good predictions and as saw above, good point estimates
- But when we do low-dimensional modeling we usually don't feel very satisfied with just point estimates
- We want confidence intervals and p-values!

High-dimensional inference

- What are the obstacles for obtaining p-values and confidence intervals?
- Highly non-standard setting when $p > n$
- the distribution of lasso-solutions, by construction, has a point-mass at 0 and this makes bootstrapping to get standard error estimates difficult

Sample-splitting

Wasserman and Roeder (2009) proposed the following approach to obtain p-values

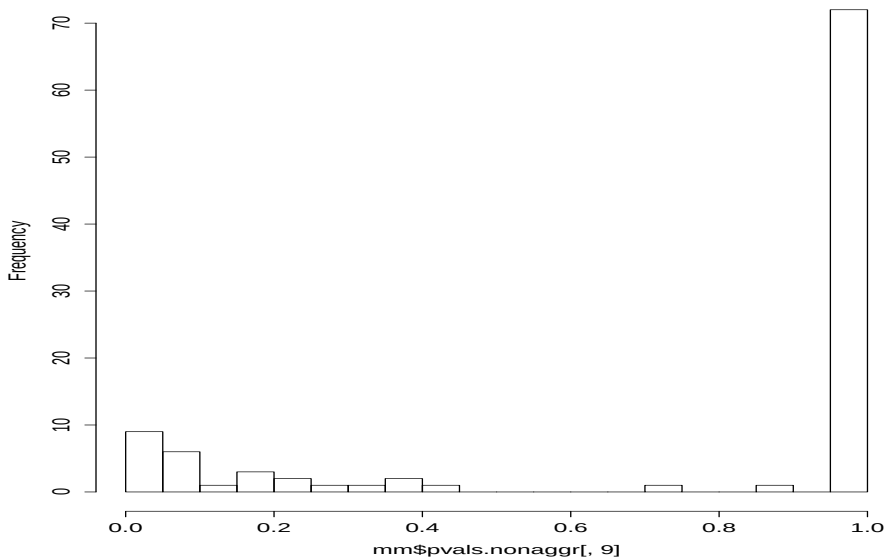
- Split the data in two sets
- Use set 1 to perform modelselection via e.g. lasso
- Use set 2 to evaluate p-values for the non-zero coefficients. This is done by running LS using only the selected variables in the model.
- For the variables not selected in set 1, set p-value to 1.

The p-values are valid because we didn't reuse the same data for selection and p-value computation.
Moreover, if we want to compute adjusted p-values that take into account multiple testing we only have to correct by the selected set of variables, not all p.

Sample-splitting

Drawback with the procedure

- Sensitive to the split so the pvalues are not reproducible
- "p-value lottery"
- Different splits leads to widely different p-values!
- Use Multi sample splitting instead `hdi()` package + de-sparsified lasso



Data representations

With big data we often need to find efficient data representations of a smaller dimension for both visualization and computation. Precursor to modeling (regression and classification)

- SVD, NMF
- SOM, MDS

SVD

SVD (singular value decomposition) is a workhorse that underpins much of the modeling we do!

- Data matrix X of dimension $n \times p$
- Before you do anything, you want to center and scale the columns of X!!!
- Otherwise the scale of individual variables dominate the representation and visualization is weird without centering
- We want to approximate the observations x_i in X by a lower-rank model

SVD

- $X = UDV^T$ where U is a $n \times p$ matrix, D is a diagonal $p \times p$ matrix and V is a $p \times p$ matrix where $U^T U = I$, $V^T V = I$
- We can also write $VX = UD$
- UD are called the *principal components*
- VX is the rotation V applied to the data X to project it onto the principal component space.
- The entries of each column in V are called *loadings* and tell you how much each original variable contribute to the make-up of the new dimension in PC space
- The leading components in V correspond to the largest values of D

SVD

- Another way of looking at SVD is building a structure from orthogonal components. To see this write

$$X = UDV^T = \sum_{j=1}^{P} d_j u_j v_j^T$$

  where u_j is a $n \times 1$ vector and v_j^T is a $1 \times p$ vector.
- Each produce $u_j v_j^T$ construct a $n \times p$ matrix representation of X
- Scaled by d_j they represent approximation of X in orthogonal directions.
- The first component is the best rank 1 approximation of X

SVD and regression

- Least squares modeling

$$\min_{\beta} ||Y - X\beta||^2$$

- The LS solution $\beta = (X'X)^{-1}X'Y$
- If we plug in $X = UDV^T$ in the above expression we get

$$\beta = (VDU'UDV')^{-1}VDU'Y =$$

$$= (VD^2V')^{-1}VDU'Y = VD^{-2}V'VDU'Y = VD^{-1}U'Y$$

- The expression $VD^{-1}U'$ is called the *pseudo-inverse* of X
- Notice then that the regression coefficients are really obtained through SVD
- Fitted values $\hat{y} = X\hat{\beta} = UDV'VD^{-1}U'Y = U(U'Y)$
- U is the ortogonal basis the spans the columns of X and regression projects onto these components

SVD and classification

- In classification with LDA we sphered the data using U
- We classified using the manahalobis distance

$$c(x) = \arg\min_c (x - \mu_c)' \Sigma^{-1} (x - \mu_c)$$

- We could write $X'X/n = \hat{\Sigma} = (VD^2V')/n$
- And so we can write

$$c(x) = \arg\min_c (V'(x - \mu_c))' D^{-2} (V'(x - \mu_c))$$

- Since the matrix D is diagonal, the sphered data is much easier to work with - just look at one "variable" at a time in this space

SVD and data representation

- SVD is a component in many methods as you saw above
- We can also use it for data exploration
- We plot the principal components $XV = UD$ for the leading components and since these preserve most of the information in X we get a dense summary of the data
- Excellent for finding groups in the data
- The loadings in V tells you in which variable set the information about X resides
- If PC1 and PC2 separates groups in the data, check which variables contribute to these (loadings in the 1st columns of V)

sparse SVD

- Like in regression, it is not always easy to see which variables contribute to the PCs
- We can look for large factor loadings....
- OR we can adapt SVD to generate sparse V where only few variables do contribute

sparse SVD

- A couple of different variants of sparse SVD have been proposed
- ScotLASS (Joliffe et al), sparse PCA (Witten et al) and sparse SVD (Zou et al) are a few
- They add an L1 penalty to the factor loadings, but how the problem then is solved is different

sparse SVD

- `PMA` package and `nsprcomp` package
- Difficult to choose how much to penalize
- but good for visualization and exploration

Non-negative matrix factorization

- What if the idea of layer summation is our key feature?
- Each layer should add some information to the representation (not correct previous layers)
- NMF:

$$X = WH, \quad W \geq 0 \quad H \geq 0$$

- X is our $n \times p$ matrix where each row is an observation and each column a feature
- W is a $n \times r$ matrix, or *basis* which gives you the coordinates for each of the n observations in the lower-dimensional space....
- ... indexed by H: a $r \times p$ matrix of coefficients, or a *codebook*.

Non-negative matrix factorization

- NMF:

$$X = WH, \quad W \geq 0 \quad H \geq 0$$

- Example from the handwritten digits: Let's say we choose to approximate the data with rank K
- H will contain K images of the same size as each original digit, illuminating important pixels that summarize the data
- W is a matrix where each row j tells you how to combine the images in H to recreate the jth digit in the data set.

Data representations

- Both SVD and NMF try to find a linear dimension reduction of the data to summarize the data well
- The difference lies in the assumed structure of the dimension reduction
- SVD creates orthogonal components (perhaps sparse)
- NMF creates component-wise non-negative coefficients and basis elements.

Non-negative matrix factorization

- Let's say we have found a rank K approximation
- We can approximate the jth observation by

$$\hat{X}_j = \sum_{l=1}^{K} W_{jl} H_{l.}$$

- Since we have a non-negative contraint on W and H this consists of adding layers together, no corrections or subtractions.
- The result of the non-negative constraint is that the coefficients in H tend to be *sparse*!!!
- Fit with alternating least squares

Revisiting PCA

- Data matrix X of dimension $n \times p$
- Assume from now that rows are centered
- SVD $X = UDV'$, PCA $X'X = VD^2V'$. Principal components UD obtained by projecting X onto V: $XV = UD$.

Revisiting PCA

- Key: want to maximize variance along orthogonal projections.
- $max_v Var(vX)$ subject to $v'v = 1$
- Lagrangian formulation $max_v v'X'Xv - \lambda(v'v - 1)$
- Take derivatives: $X'Xv - \lambda v = 0$ or equivalently $Sv = \lambda v$ - the eigenproblem

Revisiting PCA

- What if we had instead focused on XX'?
- Let's apply PCA in this system: $XX' = WLW'$: equivalently $XX'W = WL$
- Multiple by X' from the left: $(X'X)(X'W) = (X'W)L$.
- This means $X'W$ is the eigenvectors of $X'X$
- Almost, since not orthonormal: $(X'W)'(X'W) = W'(XX')W = L$ not I
- Fix by renormalizing so: $V = (X'W)L^{-1/2}$ is the eigenvectors of $X'X$ obtained from XX'

Revisiting PCA

- Why do we care?
- Alternative approach to PCA from *object distances* rather than *feature correlations*
- $X'X \simeq Cov(X)$ is the p by p structure between features p.
- $XX' \simeq distance(X)$ is the n by n distance matrix between objects

Gram matrix and the distance matrix

- Let's look at the distance matrix $d_{ij} = dist(x_i, x_j)$
- $d_{ij} = ||x_i - x_j||^2 = (x_i - x_j)'(x_i - x_j) = x_i'x_i - 2x_i'x_j + x_j'x_j$
- Now

$$XX' = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \cdots & & & \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \cdots & & & \\ x_{1p} & x_{2p} & \cdots & x_{np} \end{pmatrix} =$$

$$\begin{pmatrix} \sum_{j=1}^{p} x_{1j}^2 & \sum_{j=1}^{p} x_{1j}x_{2j} & \cdots & \sum_{j=1}^{p} x_{1j}x_{nj} \\ \sum_{j=1}^{p} x_{1j}x_{2j} & \sum_{j=1}^{p} x_{2j}^2 & \cdots & \cdot \\ \cdots & & & \\ \cdot & & \cdots & \cdot \quad \sum_{j=1}^{p} x_{nj}^2 \end{pmatrix}$$

Gram matrix and the distance matrix

- $d_{ij} = ||x_i - x_j||^2 = (x_i - x_j)'(x_i - x_j) = x_i'x_i - 2x_i'x_j + x_j'x_j$

$$XX' = \begin{pmatrix} \sum_{j=1}^{p} x_{1j}^2 & \sum_{j=1}^{p} x_{1j}x_{2j} & \cdots & \sum_{j=1}^{p} x_{1j}x_{nj} \\ \sum_{j=1}^{p} x_{1j}x_{2j} & \sum_{j=1}^{p} x_{2j}^2 & \cdots & . \\ \cdots & & & \\ . & & \cdots & . \quad \sum_{j=1}^{p} x_{nj}^2 \end{pmatrix} =$$

$$\begin{pmatrix} x_1'x_1 & x_1'x_2 & \cdots & x_1'x_n \\ x_1'x_2 & x_2'x_2 & \cdots & . \\ \cdots & & & \\ . & \cdots & . & x_n'x_n \end{pmatrix}$$

- This means we can write the distance matrix $D = (d_{ij})$ as

$$D = 1diag(XX') - 2XX' + diag(XX')1'$$

where 1 is just a column of ones.

Gram matrix and the distance matrix

- The matrix $G = XX'$ is called the Gram matrix.
- The elements of G are the dot-products of the observations $x_i' x_j =< x_i, x_j >$
- From previous slide we see that the pairwise distances are just a function of the dot-products

$$D = 1 diag(XX') - 2XX' + diag(XX')1'$$

Gram matrix and the distance matrix

- The matrix $G = XX'$ is called the Gram matrix.
- $$D = 1diag(XX') - 2XX' + diag(XX')1'$$
- Equivalently, define $H = I - \frac{1}{n}11'$ (also called the centering matrix), then
$$G = -\frac{1}{2}HD^2H$$

Multi-dimensional scaling

- MDS: only use the pairwise distances
- MDS: not restricted to 2-dim space

Multi-dimensional scaling

- We compute all the pairwise distances between objects i and j: d_{ij}
- We can be clever about using appropriate distances here depending on the variable types (`daisy` package in R)
- We want to find observations z_i in a low-dimensional space such that

$$\sum_{i \neq i'} (d_{ii'} - ||z_i - z_{i'}||)^2$$

  is small.
- We can scale the mapping distance by $d_{ii'}$ which preserved small distances better
- We can also use rank-based mapping (called non-metric scaling) - depending if subsets of data are very spread out.

Multi-dimensional scaling

- We compute all the pairwise distances between objects i and j: d_{ij}
- We want to find observations z_i in a low-dimensional space such that

$$\sum_{i \neq i'} (d_{ii'} - ||z_i - z_{i'}||)^2$$

  is small.
- How? Spectral decomposition of centered $d :_{ii'}$ and use leading eigenvectors.
- With the alternative representation of PCA through the Gram matrix and thereby the distance matrix we see that MDS is in fact equivalent to PCA if the distance used is euclidean!!!

Multi-dimensional scaling

- Metric MDS: we want to minimize $min_z \sum_{ij}(d_{ij}^x - d_{ij}^z)^2$
- We just saw the equivalence between the gram matrix and the distance matrix: minimizing D - max "correlation" or min dotproduct.
- So MDS is equivalent to $min_z \sum_{ij}(x_i'x_j - z_i'z_j)^2$
- Write in matrix form as a Trace of the full matrices

$$min_Z Tr(XX' - ZZ')^2$$

- Use spectral decomposition of each

$$min_Z Tr(WLW' - QTQ')^2 = Tr(L - W'QTQ'W)^2 =$$

$$= Tr(L - RTR')^2 = Tr(L^2 - RTR'RTR' - 2LRTR')$$

- $$min_Z Tr(WLW' - QTQ')^2 = Tr(L - W'QTQ'W)^2 =$$
$$= Tr(L - RTR')^2 = Tr(L^2 - RTR'RTR' - 2LRTR')$$
- For fixed T: minimize wrt R - solution $R = I$ and plug-in
$$min_T Tr(L^2 - T^2 - 2LTR) = Tr(L^2 - T^2)$$
- Make small by matching the leading eigenvalues of L and T and since $I = R = W'Q$ this implies $Q = W$
- So MDS - leading eigenvectors of the Gram matrix.

Multi-dimensional scaling

- We compute all the pairwise distances between objects i and j: d_{ij}
- We want to find observations z_i in a low-dimensional space such that

$$\sum_{i \neq i'} (d_{ii'} - ||z_i - z_{i'}||)^2$$

  is small.
- The point being - with MDS we can exploit this and be more flexible with the distance used!
- Define $H = I - \frac{1}{n} 11'$ (also called the centering matrix), then we can obtain a gram matrix from a distance matrix through

$$G = -\frac{1}{2} H D^2 H$$

- and then solve the eigen problem

Multi-dimensional scaling

- PCA = eigenvectors of $X'X$ (covariance of X) - scales poorly with dimensionality
- MDS = eigenvectors of XX' (gram matrix, related to pairwise distances) - scales poorly with sample size
- Key is using other distance metrics in MDS for flexibility.
- In non-metric MDS you are even more adventurous - using monotone transformations of distances, qualitative distances, ranks - usually then solved by iterative procedures.

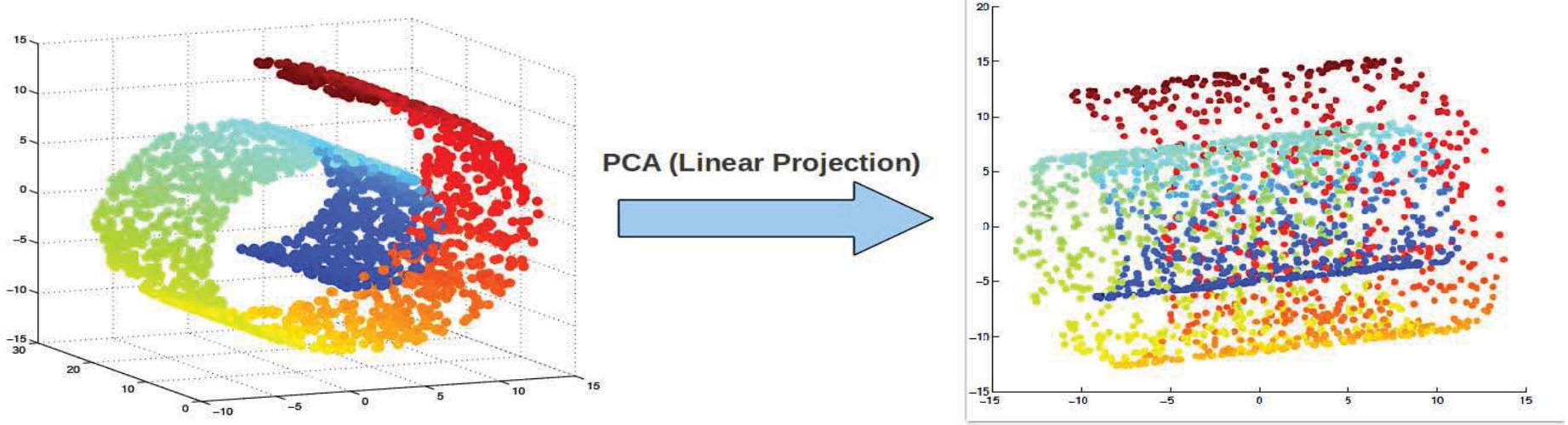
- PCA = eigenvectors of $X'X$ (covariance of X) - scales poorly with dimensionality
- or equivalently eigenvectors of XX', obtain eigenvectors as $V = (X'W)L^{-1/2}$ and PCs as XV.
- Limitations of PCA: Global method, assuming X can be well represented by a linear projection/approximations - covariance is a linear association measure
- What if we could look at nonlinear dependencies? How?
- Transform data into a $M > p$ dimensional space: $\phi(x)$ and compute covariance in this space

$$S^{\phi} = \phi(X)'\phi(X)$$

(M by M matrix instead of p by p).

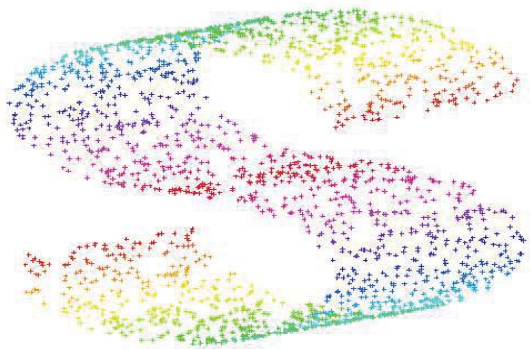
- Compute eigenvectors V: $S^{\phi}V = LV$ and project onto leading components here.

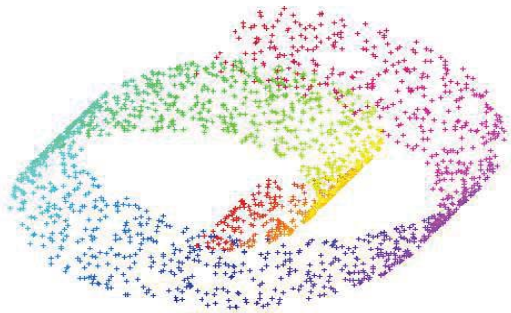
KERNELPCA



- What do we want the nonlinear transformation ϕ to do?
- Preserve local information: e.g. locally linear relationships or pairwise distance information

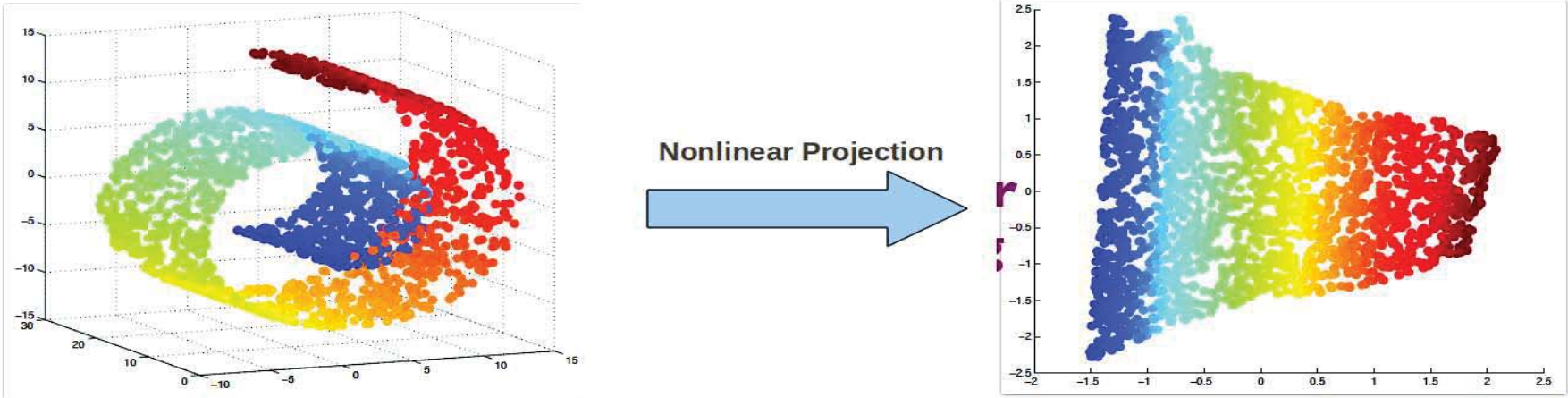
- Given: Low-dim. surface embedded **nonlinearly** in high-dim. space
  - Such a structure is called a **Manifold**





- Goal: Recover the low-dimensional surface

KERNELPCA



Figures from slides by P. Rai

- What do we want the nonlinear transformation ϕ to do?
- Preserve local information: e.g. locally linear relationships or pairwise distance information
- "Unrolling" or "unwrapping" the low dimensional structure embedded in a high dimensional space

- We would rather not have to compute the data transformation for two reasons
- Increasing dimensionality - bad
- Having to explicitly define a transformation that works
- Idea: the kernel-trick. "Easier" to define a nonlinear or local distance and we know from before there is an implicit relationship between the PCA decomposition based on structure and distance.

- We have $S^{\phi} = \frac{1}{n}\sum_i \phi(x_i)\phi(x_i)'$ as the M by M covariance matrix. (assume centered after transformation)
- PCA: $S^{\phi} v_k = \lambda_k v_k$ for components $k = 1, \cdots, M$
- Plug in: $\frac{1}{n}\sum_i \phi(x_i)(\phi(x_i)' v_k) = \lambda_k v_k$
- Eigenvectors are of the format $v_k = \sum_i a_{ki}\phi(x_i)$ and so $\frac{1}{n}\sum_i \phi(x_i)(\phi(x_i)' \sum_j a_{kj}\phi(x_j) = \lambda_k \sum_j a_{kj}\phi(x_j)$
- Multiply this by $\phi(x_l)'$ on both sides $\frac{1}{n}\sum_i phi(x_l)'\phi(x_i)(\phi(x_i)' \sum_j a_{kj}\phi(x_j) = \lambda_k \sum_j a_{kj}\phi(x_l)'\phi(x_j)$

- Multiply this by $\phi(x_l)'$ on both sides
  $\frac{1}{n}\sum_i phi(x_l)'\phi(x_i)(\phi(x_i)'\sum_j a_{kj}\phi(x_j) = \lambda_k \sum_j a_{kj}\phi(x_l)'\phi(x_j)$
- The dot products are scalars, define as $\phi(x_i)'\phi(x_j) = k(x_i, x_j)$
- and so we have
  $\frac{1}{n}\sum_i k(x_i, x_l)\sum_j a_{kj}k(x_i, x_j) = \lambda_k \sum_j a_{kj}k(x_i, x_l)$
- Define the kernel matrix K comprising all the dot products (see back at PCA via Gram) which captures pairwise distance/similariy in ϕ space

$$K^2 a_k = \lambda_k NK a_k \rightarrow K a_k = \lambda_k a_k$$

- We need to normalize the vs:
  $v_k' v_k = 1 = \sum_{i,j} a_{ki} a_{kj}\phi(x_i)'\phi(x_j) \rightarrow a_k' K a_k = 1$
- A projection onto the k-th PCs is thus

$$\phi(x)' v_k = \sum_i a_{ki} K(x, x_i)$$

- The kernel matrix has to be centered prior to this
- Use the centering matrix from above
- Popular kernels: gaussian, polynomial

- tSNE is a local extension of MDS. (Paper can be found here).
- Here we use a kernel based distance between observations i and j and interpret this as a probability

$$p_{j|i} = Gaussian - pdf(d_{ij}, \sigma_i^2) / \sum_{k \neq i} Gaussian - pdf(d_{ik}, \sigma_i^2)$$

  where σ_i is the bandwidth of the kernel around reference point i. You create a symmetric distance by taking the average of the two conditional distributions
- Even more simple if you use the same bandwidth everywhere

$$p_{ij} = Gaussian - pdf(d_{ij}, \sigma^2) / \sum_{k,l \neq k} Gaussian - pdf(d_{kl}, \sigma^2)$$

- We now try to construct a d-dimensional space y that mimics these densities where we define the pdf in this space as

tSNE

- How do we measure distance in the y-space? Natural thing would be to use gaussian densities there too (called SNE)
- In the SNE, the authors observed that the y-space got "crowded" in that slightly similar observations were forced to be very similar in the low-dimensional space
- To remedy this, tSNE uses a more long-tailed distribution to describe the densities in y-space (Cauchy distribution)

$$q_{ij} = \frac{(1 + d(y_i, y_j))^{-1}}{\sum_{k \neq i}(1 + d(y_i, y_k))^{-1}}$$

  where d is the squared euclidean distance
- We match p and q by minimizing the Kullback-Leibler distance $\sum_{i \neq j} p_{ij} \log(\frac{p_{ij}}{q_{ij}})$
- How? Gradient descent.
- So it's related to MDS, but with a different treatment of distances and a different cost function.

Isomap

- Isomap is a local version of MDS - we work with a matrix of distances between observations
- Use distances based on a shortest path in a graph connecting observations
- The graph is produced by connecting only objects that are within a certain euclidean distance of eachother, or is within a set of k nearest neighbors.
- This can capture quite local behaviour - nonlinear transformation of data
- Spectral decomposition of this matrix

Local linear embedding

- LLE - "fix" the problem with global PCA by only approximating each X by a linear combination of nearest neighbors!
- Find the L nearest neighbors of each observation
- Assume x_i can be explained by a weighted linear combination of only the neighbors

$$x_i = \sum_{j \in N_i} w_{ij} x_j, min_W \sum_i ||x_i - \sum_{j \in N_i} W_{ij} x_j||^2$$

where we normalize the weights to add to 1

- Now consider a K-dim space ($K < L$) where the same weights could approximate local behaviour

$$Z = \arg\min_Z \sum_i ||z_i - \sum_{j \in N_i} W_{ij} z_j||^2$$

where we want $Z'Z = I$ and centered Z.

Local linear embedding

- Now consider a K-dim space ($K < L$) where the same weights could approximate local behaviour

$$Z = \arg\min_{Z} \sum_{i} ||z_i - \sum_{j \in N_i} W_{ij} z_j||^2$$

  where we want $Z'Z = I$ and centered Z.
- Can rewrite the approximation error in Z-space as $\sum_{ij} M_{ij} Z_i' Z_j$ where $M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki} W_{kj}$
- The bottom eigenvectors of M solves the problem

Local linear embedding

- Assume x_i can be explained by a weighted linear combination of only the neighbors

$$x_i = \sum_{j \in N_i} w_{ij} x_j, min_W \sum_i ||x_i - \sum_{j \in N_i} W_{ij} x_j||^2$$

  where we normalize the weights to add to 1

- Lagrangian

$$\sum_i (||x_i - \sum_{jinN_i} W_{ij} x_j||^2 + \lambda_i (sum_j W_{ij} - 1))$$

  solve separately for each observation

- Now consider a K-dim space ($K < L$) where the same weights could approximate local behaviour

$$Z = \arg\min_{Z} \sum_{i} ||z_i - \sum_{j \in N_i} W_{ij} z_j||^2$$

where we want $Z'Z = I$ and centered Z.

$$\sum_{i} ||z_i - \sum_{j \in N_i} W_{ij} z_j||^2 - \sum_{ab} \lambda_{ab} (\sum_{i} z_{ia} z_{ib} - \delta ab)$$

which after some manipulation can be written as

$$(I - W)'(I - W)Z = Z\Lambda$$

an eigenvalue problem (details in the paper).

Global/Local

- Global: PCA, SVD, MDS, NMF
- Local: kPCA, isomap, tSNE, LLE
- many many more.... I have posted some review papers.
- Of course, be careful about applying to real data! "Overfitting", noisy data, big/small n, big/small p, mixed data types...

Clustering

Explorative analysis - finding groups in data.
This is a more difficult task than classification since the goal is rather subjective - what is group?

What defines a group is up to you to choose, e.g. by defining an object-object similarity measure or distance.
The most commonly used distance measure is euclidean distance.
However, other distances may be more appropriate to use for some data sets, e.g. matching-metrics for categorical data or correlation-based similarities for curve data or when relative differences between features are more interesting than absolute levels.

Goal of Clustering

1. Minimize within-cluster distances.
   - C denotes a *partition* that puts labels $\{1, \cdots, K\}$ on objects. $C(i)$ is the label for observation i.
   - We want the within-cluster distances to be small: $W(C) = \sum_{k=1}^{K} \sum_{C(i)=C(j)=k} d_{ij}$, where d_{ij} is the distance between objects i and j.
2. Maximize between-cluster distances.
   - Maximize $B(C) = \sum_{k=1}^{K} \sum_{C(i)=k, C(j)\neq k} d_{ij}$

Turns out the total sum of distances is $W(C) + B(C)$ so the two goals are equivalent. (But if you only consider distances to some clusters instead of all, it's not the same.)

How to generate a partition

kmeans is a very popular method and has been around for a long time. It is very simple and fast and results in partitions that minimize the within-cluster distances.
In addition, kmeans doesn't try to take cluster shape into account and tends to find spherical groups (cf. nearest-centroid classifier).
How many clusters? You can track how much the within-cluster distances decrease as a function of the number of clusters. Once you start adding more clusters that the data "supports" there is very little decrease in $W(C)$ since you are forcing kmeans to split close observations into smaller groups. You should look for a point where the $W(C)$ levels off as the number of clusters grow (see demo code).

k-medoids, PAM

PAM, partition around medoids, is an alternative to kmeans that is more robust and allows for extensions to non-euclidean distances. Instead of using centroids (cluster means) to summarize a group, we use an observation = a medoid. Medoid is a multivariate generalization of a median. Some things to note about PAM

- The entire algorithm uses only object-object distances d_{ij}
- No new distances need to be computed (to centroids etc since medoids are already part of the set of observations).
- Input to PAM can thus be any kind of pairwise distance matrix! Lots of choices possible.

Hierarchical clustering

Hierarchical clustering is very popular since it is simple, intuitive and comes with a nice graphical display. Like PAM, it takes pairwise distances as input which makes it rather flexible.
In contrast to PAM and kmeans it constructs clusters "bottom-up", i.e. building clusters by joining observations together as opposed to splitting the data into groups ("top-down").

Hierarchical clustering

1. Start with all the observations as their own clusters, $g_1, g_2, \cdots, g_n$, each cluster of size 1.
2. Join the pair of clusters g_i and g_j that are the closest together
3. Keep on joining clusters pairs until all observations are in one big clusters of size n.

Modelbased clustering

So far we have looked at nonparametric cluster methods - clusters are defined through a distance metric and a construction algorithm/criterion. We have noted that clustering is a difficult problem because these choices are subjective.
Parametric, or modelbased clustering, takes clustering into a familiar statistical modeling framework where we can say something about the goodness-of-fit of clusters. It is a statistical problem that can be objectively analyzed BUT of course relies on a modeling assumption that is a subjective choice nonetheless.

So on the plus-side, we make clear modeling assumption and can validate them and analyze/draw inference from results.
On the minus-side, our modeling assumption can be too strict, not flexible enough to capture groupings in the data.
The most commonly used modeling assumption in modelbased clustering is that each group is multivariate normally distributed, that is, groups are ellipsoid blobs in x-space.

EM-iterations

- If we knew the labels, we could estimate the parameters of each cluster easily just like we did in discriminant analysis.
- If we knew the model parameters, we could allocate observations to each cluster using the posterior probability approach, just like in discriminant analysis.

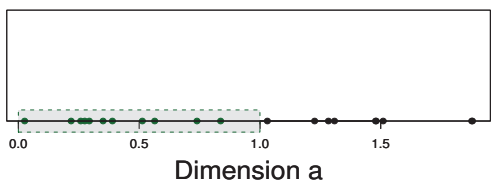
This iterative process is the EM approach to model fitting and is a method used to solve complex estimation problems that would be easy to solve with some additional information (as done in each step or the iterative procedure).

As in all model-based methods, the likelihood cannot be used to select the model (number of clusters) as the likelihood is always increased by adding more and more model parameters to the description of the data - i.e. using the likelihood to select the number of clusters would just lead you to choose the largest number of clusters you try.
However, as we are in a standard modeling setting we can use the off-the-shelf model selection criteria that you may be familiar with from regression. The most commonly used criterion in mixture modeling is the Bayesian information criterion, BIC.

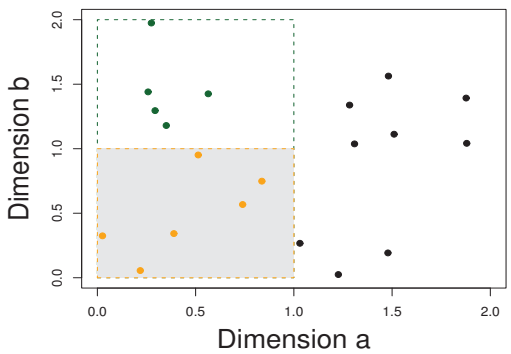
This idea of simplifying the cluster shapes, sizes and correlations (orientations) was the basis for the Mclust procedure of Raftery et al (2006). This modelbased clustering method has been implemented in a very easy-to-use R-package (`mclust()`).
Other Big Data extensions regularize the means and covariances of each cluster! Regularization methods here as well!

High-dimensional clustering

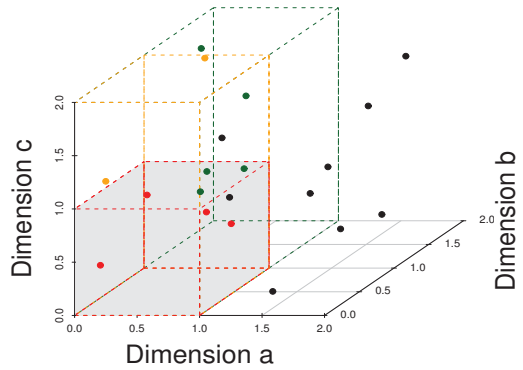
What goes wrong when the data is high-dimensional?



(a) 11 Objects in One Unit Bin



(b) 6 Objects in One Unit Bin

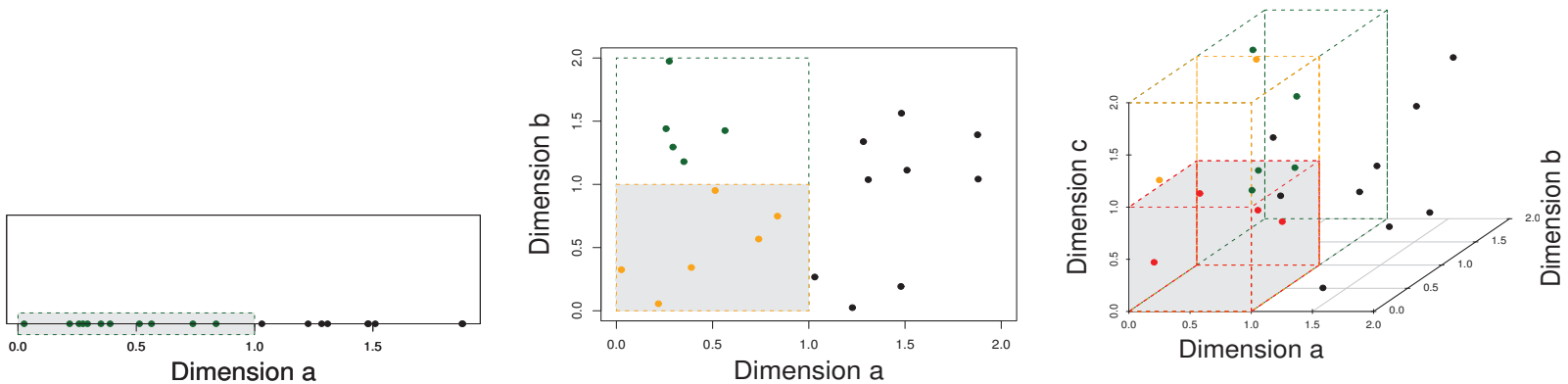


(c) 4 Objects in One Unit Bin

Figure 1: The *curse of dimensionality*. Data in only one dimension is relatively tightly packed. Adding a dimension stretches the points across that dimension, pushing them further apart. Additional dimensions spreads the data even further making high dimensional data extremely sparse.

High-dimensional clustering

The notion of similar and dissimilar brakes down - everyone is far apart in high-dimensional space!
Clustering is all about distances - and the concept of relative distance brakes down.



(a) 11 Objects in One Unit Bin (b) 6 Objects in One Unit Bin (c) 4 Objects in One Unit Bin

Figure 1: The *curse of dimensionality*. Data in only one dimension is relatively tightly packed. Adding a dimension stretches the points across that dimension, pushing them further apart. Additional dimensions spreads the data even further making high dimensional data extremely sparse.

High-dimensional clustering

What to do?

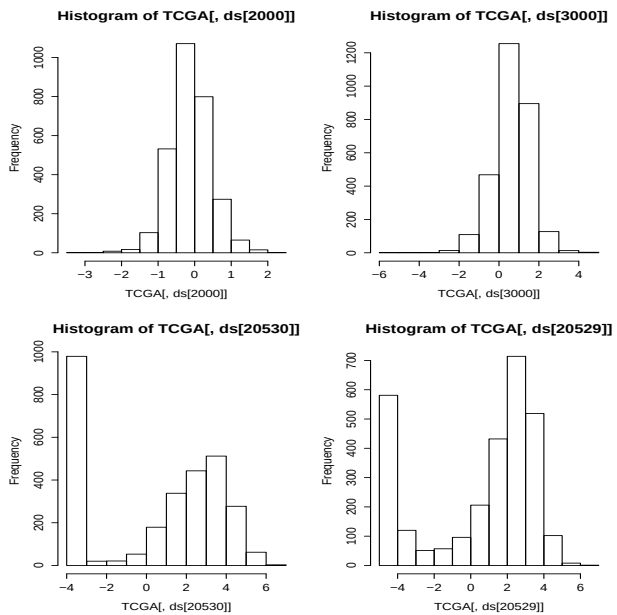
- Feature selection
- Feature transformation

Feature selection

- Take the most variable features.
- The idea is that large spread is due to cluster separation
- Caution: this is a bad idea if features are measured at different scales!

Feature selection

- An alternative is to think that a cluster feature should have a clear multi-modal distribution where each "hump" in the distribution corresponds to a cluster
- Screen features by testing for unimodality (Hartigan's dip-test).
- Keep features with the largest test statistic against unimodality



Feature transformation

- We can also transform the data - projecting onto a lower-dimensional space
- We want to ensure we retain as much information as possible
- PCA to the rescue!
- Keep principal components corresponding to the largest eigenvalues
- Or use other dimension reduction techniques discussed above!

Dimension reduction and clustering

- Careful! Check how sensitive the results are to your screening
- both the type of screening and
- how aggressively you screen

Dimension reduction and classification

- An "easier" filtering problem since you can test feature-class associations
- BUT careful about imbalanced data, correlated filter statistics, etc
- Dimension reduction - not guaranteed to retain class information!
- Other alternatives: supervised dimension reduction (PLS), effective dimensionality (classes are described with low-rank covariance structures (HDDA and similar)).

Consensus clustering

- Any method that comprises many steps is subject to instability since each step is a source of error
- How many features, how many eigenvalues?
- In addition, many clustering methods are quite sensitive to small data perturbations

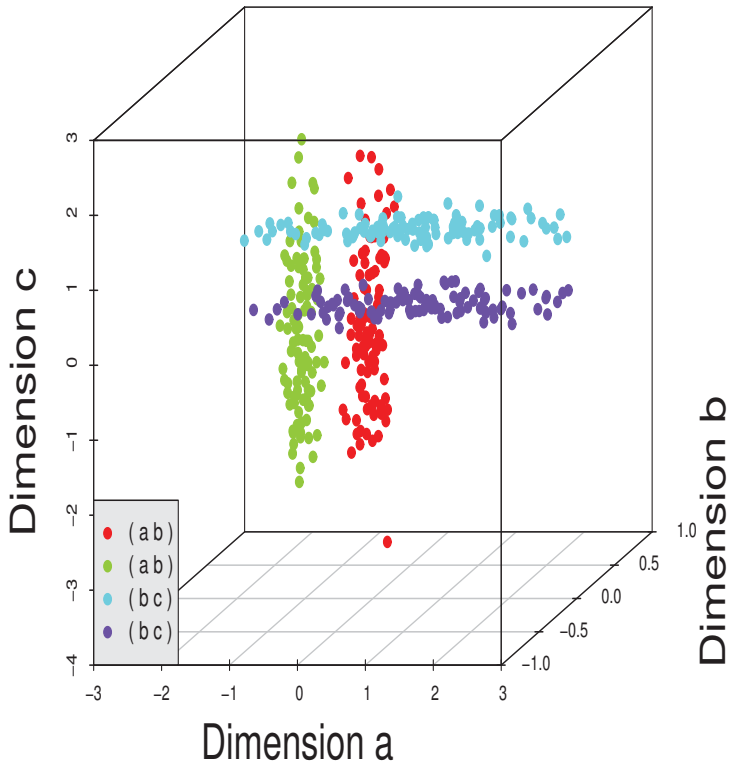
Consensus clustering

- If you can do things once, you can do it 100 times!
- Add some randomness to the procedure and run it many times
- Retain clusters that are *stable* across multiple runs!

Subspace clustering

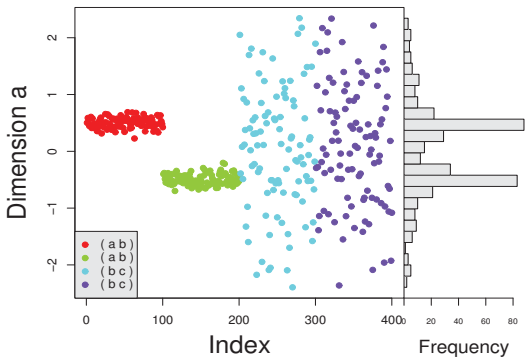
- Another method for dealing with high-dimensional data
- Assume each cluster only "lives" in a subspace (subset) of dimensions
- If we knew which subspace we could adjust how we compute distances and circumvent the COD (curse of dimensionality)

Subspace clustering

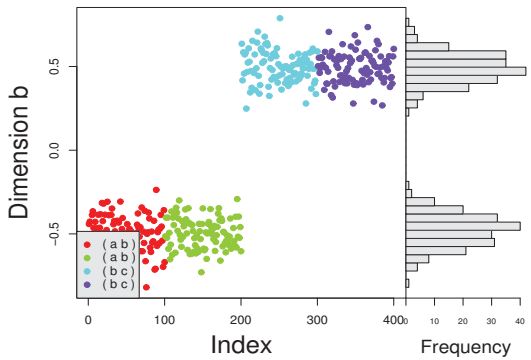


- Here are some nice figures from sigkdd review paper (see class home page) of Parsons, Hague and Liu
- 4 clusters that live in different subspaces

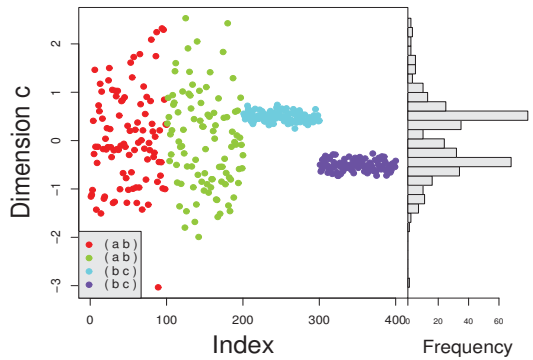
Subspace clustering



(a) Dimension a

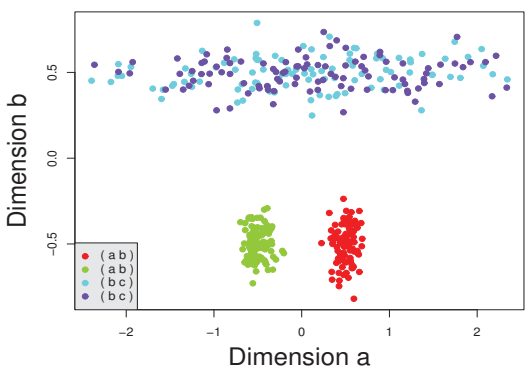


(b) Dimension b

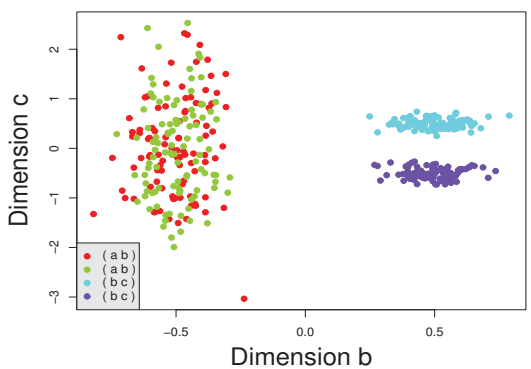


(c) Dimension c

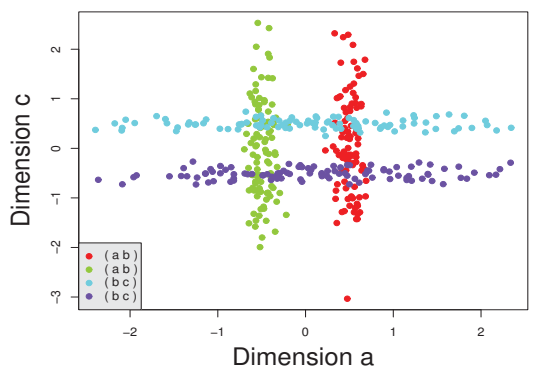
Figure 3: Sample data plotted in one dimension, with histogram. While some clustering can be seen, points from multiple clusters are grouped together in each of the three dimensions.



(a) Dims a & b

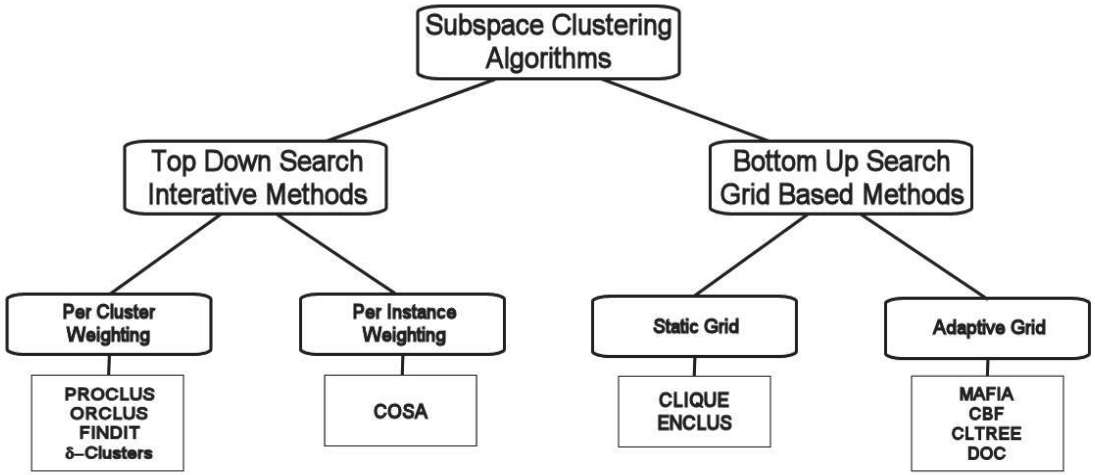


(b) Dims b & c



(c) Dims a & c

Subspace clustering

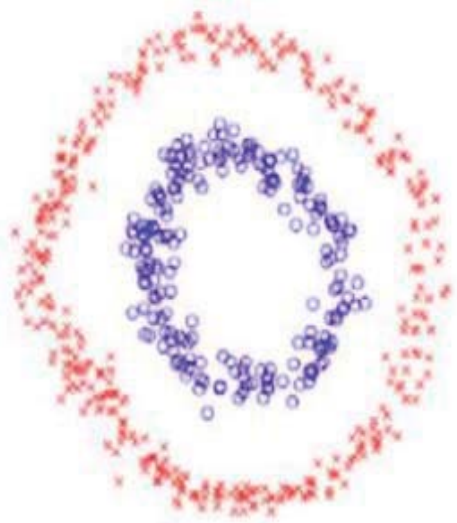


- In subspace clustering there are also two main approaches
- Bottom-up/Grid-based
- Top-down search

Spectral clustering

- Compactness, e.g., k-means, mixture models
- Connectivity, e.g., spectral clustering

Compactness



Connectivity

https://charlesmartin14.wordpress.com/2012/10/09/spectral-clustering/

- Most clustering methods are geared at finding dense, compact regions
- What if clusters are more complex than that?

Spectral clustering

Similarity graphs

- A similarity measure between observational pairs can be illustrated with a graph
- The length of an edge between objects inversely proportional to the similarity
- If we threshold similarities that are small we get a graph where only some observations are connected
- Graph-partitioning: which edges should we cut to form good clusters? Clearly the ones with low similarity.

Spectral clustering

- Turns out: if there are k connected components in your graph (clusters) then there are k zero-eigenvalues of $L = D - W$, where D is the degree matrix and W the adjacency matrix.
- and the corresponding eigenvectors can be used to find the clusters using e.g. kmeans.

Big n!

- When the sample size is large, there's a couple of things we need to be concerned about
- p-values become "meaningless" - simply reflecting that all models are approximations of the real world
- Computations can become impossible or slow, even for simple statistical tasks

Big n statistics

- Most methods bear a strong resemblance to stuff you're already familiar with
- Cross-validation, subsampling (bagging, RF) and bootstrap

Big n statistics

3 main approaches

- Subsampling
- Divide and Conquer/Split and Merge/Divide and Recombine
- Online updating

Bag of Little Bootstraps

- We draw s subsets of data of size $m < n$
- For each of the s subsets, draw r samples of size n
- Obtain point estimate and e.g. CIs from the r bootstraps
- Finally, combine the results across the s subsets

Divide and Conquer

- Idea is to split the data into K chunks
- Estimate your model parameters on each chunk separately
- Combine the estimates into a final estimate

Divide and Conquer

- For chunk k of data we solve

$$M_k(\theta) = \sum_{i \in k} \Psi(y_i, \theta) = 0$$

- Denote the estimate $\hat{\theta}_{n,k}$
- We compute

$$A_k = -\sum_{i \in k} \frac{\partial \Psi(y_i, \theta)}{\partial \theta}|_{\hat{\theta}_{n,k}}$$

- and *linearlize the scoring equation* (1st order Taylor expansion)

$$M_k(\theta) \simeq A_k(\theta - \hat{\theta}_{n,k})$$

- The approximate solution to the global scoring equation is

$$\sum_k M_k(\theta) = \sum_k A_k(\theta - \hat{\theta}_{n,k}) = 0$$

- which can be solved as

$$\hat{\theta} = (\sum_k A_k)^{-1}(\sum_k A_k \hat{\theta}_{n,k})$$

Divide and Conquer

- The solution for the nonlinear problem now looks very similar to the regression example

$$\hat{\theta} = (\sum_k A_k)^{-1}(\sum_k A_k \hat{\theta}_{n,k})$$

- AND, if you recall what you know about MLE....

$$A_k = -\sum_{i \in k} \frac{\partial \Psi(y_i, \theta)}{\partial \theta}|_{\hat{\theta}_{n,k}}$$

- The expected value of A_k if called the *Information matrix* and its inverse is the asymptotic variance of the MLE!!!
- So, the solution above is also a kind of weighted average of estimates with weights inversely proportional to the estimation variance!!!

An Online method: Maximum MEAN Likelihood

The ALGORITHM

- Initialize the parameter estimate vector θ_0
- For $t = 1, \cdots, T$, draw a sample of size m from the full data, without replacement
- Update each parameter estimate as

$$\theta_j^{t+1} = \theta_j^t + a_{t+1} \nabla_{\theta_j} \log f(\theta^t, y_t)$$

- *That is, use the gradient vector based on new data and the previous estimate of the parameters!!!*
- There is a check-point before the estimate is accepted, the new estimate can't be too far off the previous one (technical details in the paper beyond the scope of this class).

Online methods

- Ensemble methods are very popular!
- Why? Because they lend themselves to easy adaptation - weights on ensemble members, replacing or update ensemble members.
- Key: keep only summary statistics (model parameters, performance measures, microclusters) from each batch of data and adjust methods to work from these
- Can be used on data with concept drift
- Still.... not an easy problem since adaptive = flexibility/model selection

Take-home message!

Big p!

- Use data representations to explore!
- Transformation or feature selection. Regularization methods are very popular!
- Still, how much to regularize is difficult to choose. Also, it's not magic! If you have very strong correlations between "relevant" and "irrelevant" features, feature selection in extremely difficult!
- Do many times! Look for reproducibility. Variable importance like RF also useful
- Regularization methods for regression, classification and clustering!
- Nonlinear data representations - but careful about "overfitting"

Take-home message!

Classification and Predictive modeling

- Train and Test!!!
- If you can do things once you can do it 100 times. Ensemble methods or to check reproducibility.
- The answer in statistics is always "it depends" - there is no universally best method - it's case by case, dataset specific
- In general though: more data needed to train complex models

Take-home message!

Clustering

- A VERY difficult problem!
- Very subjective - careful to not read to much into this

Take-home message!

Big n!

- Subsampling (either "thinning", batch computation or streaming)!
- New R tools that can scale! (but are poorly documented...)
- Lot's happening here - some methods development for base model (VFDT), some about updating ensembles or members (Adaptive HT, ASHT, Accuracy weighted ensembles).
- Is your data stationary or not?